Technical Debt Principal Assessment through Structural Metrics

Makrina Kosti[1], Apostolos Ampatzoglou[1], Alexander Chatzigeorgiou[2], Georgios Pallas[1], Ioannis Stamelos[1], Lefteris Angelis[1]
[1] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece
[2] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
mkosti@csd.auth.gr, apostolos.ampatzoglou@gmail.com, achat@uom.gr, gpall@auth.gr, stamelos@csd.auth.gr, lef@csd.auth.gr

***Abstract*—One of the first steps towards the effective Technical Debt (TD) management is the quantification and continuous monitoring of the TD principal. In the current state-of-research and practice the most common ways to assess TD principal are the use of: (a) structural proxies—i.e., most commonly through quality metrics; and (b) monetized proxies—i.e., most commonly through the use of the SQALE (Software Quality Assessment based on Lifecycle Expectations) method. Although both approaches have merit, they seem to rely on different viewpoints of TD and their levels of agreement have not been evaluated so far. Therefore, in this paper, we empirically explore this relation by analyzing data obtained from 20 open source software projects and build a regression model that establishes a relationship between them. The results of the study suggest that a model of seven structural metrics, quantifying different aspects of quality (i.e., coupling, cohesion, complexity, size, and inheritance) can accurately estimate TD principal as appraised by SonarQube. The results of this case study are useful to both academia and industry. In particular, academia can gain knowledge on: (a) the reliability and agreement of TD principal assessment methods and (b) the structural characteristics of software that contribute to the accumulation of TD, whereas practitioners are provided with an alternative evaluation model with reduced number of parameters that can accurately assess TD, through traditional software quality metrics and tools.**

***Keywords*—technical debt; prediction model; case study;**

I. Introduction

Technical Debt (TD) monitoring and identification are among the most important activities of TD management [1]. To effectively monitor and identify TD items, the two pillars of the TD metaphor (i.e., principal and interest) need to be quantified, or at least assessed. Although the quantification of interest is an open research topic that is far from trivial [2], in the state-of-the-art one can identify several approaches for quantifying or assessing TD principal. Such approaches can be divided into two main categories: (a) ***methods that monetize the amount of TD principal*** (e.g., Software Quality Assessment based on Lifecycle Expectations—SQALE [3]), and (b) ***methods that provide proxies of TD principal through structural metrics*** (e.g., [4]). On the one hand, the state-of-the-art approaches for monetizing TD are usually based upon rule violations of different levels of granularity (i.e., ranging from architecture to source code) and criticality (i.e., ranging from warnings to critical violations), whereas at the same time omit some useful structural indicators (e.g., lack of cohesion, change proneness, etc.). On the other hand, assessments through structural quality metrics are not facilitating the metaphor of TD in a holistic manner (since its financial aspect is not considered) and are useful mostly for comparison purposes. In particular, the metric-based assessments cannot be used for classifying an artifact as low-TD or high-TD, in the sense that in the state-of-research and -practice there are no well-established thresholds for these metric scores.

In this paper, we investigate the relationship between these approaches. Specifically, we investigate if the monetized TD principal assessments can be subsumed by metric scores. The contribution of this work is two-fold: (a) it acts as a way to assess the reliability of the existing assessment approaches, in the sense that we check their agreement; and (b) it provides a small set of metrics that are easier to be calculated than SQALE and in practice they are usually calculated as part of the quality assessment process of many industries. To achieve this goal, we performed a case study on 20 open-source Java projects and built a predictive model that quantifies TD principal based on a set of structural metrics scores. The rest of the paper is organized as follows: in Section II we present related work, in Section III, we overview the case study design, whereas in Section IV we present its results. The results are interpreted and discussed in Section V, and threats to validity are provided in Section VI.

II. Related Work

In this section, we discuss the related work on how the technical debt principal is monetized. According to Alves et al. [4], the *principal* of technical debt is related to the effort and accompanying cost to eliminate the debt from a given system or artifact. Current software analysis tools offer estimates of TD principal based on counts of detectable violations. Curtis et al. [5] note that three parameters are required for such estimates, namely the number of should-fix violations in an application, the hours to fix each violation, and the cost of labor. A similar approach is adopted by the SQALE method proposed by J. L. Letouzey [3], in which a remediation index is obtained for requirements of an applicable Quality Model. For example, for a requirement stating that all files should have at least 70% code coverage, the corresponding remediation action is to write additional tests. A remediation function maps effort to each action, for example, 20 minutes per uncovered line of code. Finally, for each artifact, the remediation index relating to all the characteristics of the Quality Model is obtained by adding all remediation indices linked to all quality requirements. The resulting SQALE Quality Index is considered to represent the principal of the TD for the assessed source code.

III. Case Study Design

The goal of our study is to investigate the relation between metric-based and monetized TD assessments, and build a model based on metrics that can predict the monetized assessments. Our study is designed based on the guidelines of Runeson et al. [13].

Research Objectives and Research Questions. The main goal of this study is to model the relation between TD principal and a variety of quality metrics. Since a plethora of structural quality metrics is available, we want to accurately estimate TD by using only a pivotal subset of them. Therefore, we focus this study in two particular directions: (a) investigate the relation of each metric with TD principal individually, and (b) investigate the relation of metrics combination with TD principal. In this respect, we have set the following research question: "*Is there a way to accurately relate TD principal to other quality covariates via a specified link function?*"

Case Selection and unit analysis. As subjects for our analysis we used 20 OSS (Open Source Software) projects. The selected OSS projects have already been used in a similar context [14]. The selection criteria for these projects are discussed in detail by Arvanitou et al. [14], accompanied by a justification on how they guarantee the collection of a high-quality, well-known and established set of cases. In a nutshell, the selected projects are among the most popular projects in *Sourceforge.net* and are distinguished by the OSS community. Specifically: (a) all projects have more than ***20 official releases*** on their history background; (b) each case contains more than ***300 classes***, which is an index of complexity, and (c) due to a limitation of the tool used to quantify the structural quality metrics the selected projects had to be ***developed in Java***.

Data Collection and Data-Preprocessing. In this paper we have selected to explore the power of structural quality metrics to assess TD principal. In this process we were interested in selecting metrics that are maintainability predictors. Thus, two metric suites proposed by Li and Henry [9] and Bansyia et al [8] have been used (see Table I). To automatically extract the scores for each metric we used the Percerons Client [14], [15], [16]. *Percerons Client* calculates metric scores by parsing source code structures, while some metrics could also be calculated based on design artifacts (i.e. class diagrams). However, to serve the needs of our study, we make the assumption that design artifacts are produced as detailed as required in order to proceed with the implementation phase and that the source code implementation follows the expected design, without any drift. Inferring that the previous assumptions hold, metrics calculated at source code level will correspond to those calculated at the detailed-design level. These assumptions induct threats to our study and are analyzed in Section VI.

TABLE I. Metrics Potentially Associated with TD Princaipal

Suite	Metric	Description	Quality Attribute
Li & Henry [9] & CK [7] metrics	DIT	*Depth of Inheritence Tree*: Inheritence level number, 0 for the root class.	Inheritance
	NOCC	*Number of Children Classes*: Number of direct sub-classes that the class has.	Inheritance
	CBO	*Coupling Between Objects:* Number of inwards and outwards dependencies of a class	Coupling
	NOM	*Number of Methods*: Number of methods in the class.	Size
	MPC	*Message Passing Coupling*: Number of send statements defined in the class.	Coupling
	DAC	*Data Abstraction Coupling*: Number of abstract types defined in the class.	Coupling
	RFC	*Response For a Class*: Number of local methods plus the number of methods called by local methods in the class.	Coupling
	LCOM	*Lack of Cohesion of Methods*: Number of disjoint sets of methods (number of sets of methods that do not interact with each other), in the class.	Cohesion
	WMPC	*Weighted Method per Class*: Average cyclomatic complexity of all methods in the class.	Complexity
	SIZE1	*Lines of Code*: Number of semicolons in the class.	Size
	SIZE2	*Number of Properties*: Number of attributes and methods in the class	Size
QMOOD [8]	NOH	*Number of Hierarchies*: Number of class hierarchies in the design.	Inheritance
	ANA	*Average Number of Ancestors*: Average number of classes from which a class inherits information.	Inheritance
	DAM	*Data Access Metric*: Ratio of the number of private (protected) attributes to the total number of attributes.	Encapsulation
	DCC	*Direct Class Coupling*: Number of other classes that the class is directly related to (by attribute declarations and message passing).	Coupling
	CAM	*Cohesion Among Methods*: Sum of the intersection of a method parameters with the maximum independent set of all parameter types in the class.	Cohesion
	MOA	*Measure of Aggregation*: Number of data declarations whose types are user classes.	Coupling
	MFA	*Measure of Functional Abstraction*: Ratio of the number of methods inherited by a class to the total number of methods accessible by methods.	Inheritance
	CIS	*Class Interface Size*: Number of public methods	Size
	NOP	*Number of Polymorphic Methods*: Number of methods that can exhibit polymorphic behavior	Complexity

Regarding the quantification of TD, we used version 6.3 of *SonarQube*[1]—also known as Sonar—without further configuration and according to its default status. Sonar is an OSS platform for the continuous inspection of code quality. The platform supports a plethora of programming languages and it can offer detailed reports regarding duplicated code, coding standards, unit tests etc. The completion of data collection resulted in a dataset of 10,029 cases / units of analysis (i.e., classes) and 22 variables, 21 representing the quality metrics and one representing the monetized TD assessment (see Table I). While "cleaning-up" the dataset from extreme values and outliers (visually, using boxplots and statistically by checking iteration after iteration the cases with residuals with three or more standard deviations from the mean) approximately 1,100 cases were excluded. Descriptive statistics of the dataset are presented in Table II.

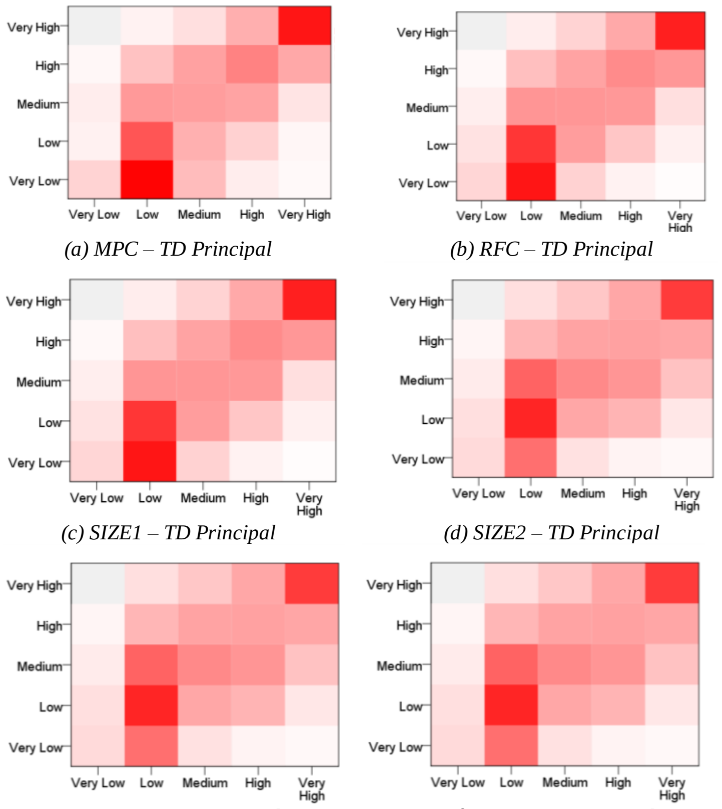
TABLE II. METRICS' DESCRIPTIVES

Metric	Mean	s.dev.	Metric	Mean	s.dev.
TD Principal	57.2	96.3	ANA	1.3	2.0
DIT	2.0	1.3	DAM	301.8	459.6
NOCC	0.7	4.2	DCC	5.5	7.2
MPC	39.4	71.6	CAM	77.3	382.4
DAC	0.3	0.8	MOA	0.6	1.4
RFC	35.6	41.8	MFA	0.015	0.1
LCOM	76.5	415.8	CIS	6.4	10.1
WMPC	8.4	11.5	NOP	1.3	5.99
SIZE1	50.1	64.8	NOM	8.4	11.5
SIZE2	11.8	15.8	CBO	11.4	19.6
NOH	0.1	0.3			

Data Analysis. To answer our research question we performed a two-step analysis process. Initially, we investigated the relationship between the monetized TD assessment and the structural quality metrics in order to reveal the quality metrics that are mostly correlated to the TD principal. More specifically, we intend to quantity the rank correlation (statistical dependence between the ranking of these variables), and a well-known measure to do this is the Spearman's correlation coefficient, (r_s). We formally choose to report the non-parametric test as our variables were not normally distributed. The equivalent Pearson correlations though were also in line with the Spearman ones and the size of our sample explains this accordance [17]. Next, we employed multiple linear regression analysis. To this end we applied the IBM®[2] statistics' stepwise regression (SR) procedure. The stepwise method is useful when it comes to selecting the optimal predictors (among a large number of candidates) that can achieve the best estimation ability of the model. In a nutshell, it is a technique of fitting regression models in which the choice of predictive variables is performed in an automated way. This way, the regression model is constantly re-assessed as it is checked if any predictors can be excluded. SR is a parametric procedure assuming that the data is normally distributed. This constraint though can be ignored as we deal with a sample consisting of hundreds of observations [17].

IV. RESULTS

In this section, we present the results of our case study and the discussion of these results takes place in Section IV. As a first step towards our analysis, we first explored the correlations among the independent variables of our prediction model (i.e., the structural metrics). The interrelations of the structural quality metrics showed high and statistically significant correlations (r_s) between specific metrics[3]. These relations can be intuitively explained based on one or both of the following interpretations: (a) relations of metrics of the same quality attribute (e.g., RFC and MPC are both coupling), (b) relation of all metrics to size (e.g., the largest the number of classes—i.e., SIZE1—the largest the number of dependencies and local methods—e.g., RFC).



(a) *MPC – TD Principal* (b) *RFC – TD Principal*

(c) *SIZE1 – TD Principal* (d) *SIZE2 – TD Principal*

(e) *WMPC – TD Principal* (f) *NOM – TD Principal*

Fig. 1. Correlation heatmaps of the most correlated to TD quality metrics

Regarding the interrelations of the structural metrics to the monetized TD Principal, we highlight the following, based on the Spearman's correlation coefficient: MPC (r_s=.72), RFC (r_s=.68), SIZE1 (r_s=.67), SIZE2 (r_s=.56), WMPC (r_s=.504) and NOM (r_s=.504). Therefore, we should expect some of these predictors to be able to explain a significant amount of the variance of TD principal. All correlation values were positive, except for NOH, NOP, MFA, NOCC and DIT, which were

[1] https://docs.sonarqube.org/display/SONAR/Documentation

[2] http://www-01.ibm.com/support/docview.wss?uid=swg27043946

[3] MPC – RFC, RFC – (WMPC, SIZE1, SIZE2, NOM), WMPC – (CIS, SIZE1, SIZE2), CIS – (WMPC, SIZE2, NOM), SIZE1 – (SIZE2, NOM), NOM – (SIZE1, SIZE2, RFC), DIT – ANA

negatively correlated to TD principal. The sign of these relations can be interpreted by the positive effect that polymorphism and inheritance have on software quality, in the sense that they are expected to provide an effective way to handle software complexity. At this point it is necessary to clarify that TD principal is an inverse quality indicator, i.e., modules with high TD principal are considered of low quality.

To visualize the relations between metrics and TD principal we provide a set of heat maps for the top-6 predictors (see above Figures 1.a - 1.f). For all figures, the horizontal axis represents the TD principal, whereas the vertical ones the values of the metric. The categorical versions of variables have been obtained with using the quintiles of each variable. To facilitate the readability of the figures, the scale in the horizontal axis is the same in all heat maps, as follows: Very Low TD [1 to 4 minutes to resolve], Low TD [5 to 10 minutes to resolve], Medium TD [11 to 33 minutes to resolve], High TD [34 to 87 minutes to resolve], and Very High TD [88 to 1803 minutes to resolve]. We remind the reader that the needed refactoring time is calculated at class level. From Figure 1, we can observe that the dark-shaded cells are the diagonal ones, e.g., high TD classes are presenting high levels of the metric scores.

The results of the regression (see Table III) indicated that the predictors can explain 74% of the dependent variable variance ($R^2 = 0.74$, $R^2_{Adjusted} = 0.74$, and $p << 0.01$). Multicollinearity issues were evaluated either by examining the correlation matrices or by checking the variance inflation factor (VIF). VIF indicates weather a predictor has a strong linear relationship with other predictors participating in the model [18].

TABLE III. REGRESSION MODEL COEFFICIENTS

Model	**Coefficients**	**T**	***p***
(Constant)	15.054	14.032	<0.001
MPC	0.802	65.982	<0.001
SIZE1	0.627	31.643	<0.001
CIS	-3.294	-28.359	<0.001
NOP	2.642	20.238	<0.001
MOA	5.584	13.127	<0.001
DIT	-4.037	-10.188	<0.001
LCOM	0.017	8.815	<0.001

Based on the correlation matrices, we ascertain that our model does not have predictors which are highly correlated to the other ones (a threshold of 0.8 was adopted). The third column of Table VII represents the *t* statistic, a measure whose magnitude shows the impact of a predictor to the model. Therefore, (a) we built a model with 7 estimators that can predict the TD principal with an accuracy of 74%; and (b) among these metrics the three most influential ones are: MPC, with the highest impact in the model, followed by SIZE1 and CIS.

V. DISCUSSION

In this section, we discuss the results of this study, under two perspectives: (a) we interpret the results and (b) we provide some useful implications to researchers and practitioners.

Interpretation of Results. The results of our study validated the intuition that TD principal can be sufficiently assessed through a limited set structural metrics. In particular, we have been able to build a seven metrics model that can capture 74% of the variability of TD amount (expressed in effort—i.e., minutes of development). In particular, the model consists of:

- *Two coupling metrics* (MPC and MOA). As expected both coupling metrics are positively related to TD principal, i.e., the higher the coupling the higher the amount of TD that exists in a module. The fact that MPC (i.e., the number of distinct method calls from one class to another) is the most important predictor of TD principal can be attributed to the fact that MPC captures: (i) the difficulty to apply changes in a given module due to the amount of dependencies, and (ii) the possibility of a class to change due to ripple effects, in the sense that it is a proxy of the strength of the dependency.
- *Two size metrics* (CIS and SIZE1). Based on the findings of this study larger systems (with more lines of code, attributes, and methods) are accumulating more TD. Despite the fact that this finding is a self-explanatory one, the negative relationship of CIS to TD amount is an interesting one. In particular, the direction of the relationship suggests that modules with a large public interface (i.e., methods) are less prone to accumulate TD. This fact can be attributed to the modularity of responsibilities offered by different methods, e.g., long methods have already been split into smaller more modular ones.
- *One cohesion metric* (LCOM). Lack of cohesion is as expected positively related to TD principal. This is an expected outcome in the sense that: (i) high cohesion is one of the most important principles of object-orientation, and (ii) lack of cohesion directly implies the existence of the large class “bad smells”, which urges for the application of well-known refactoring.
- *One polymorphism metric* (NOP) and *one inheritance metric* (DIT). These metrics are the only with a negative correlation to TD principal. This finding can be explained by the fact that the efficient of application of object-oriented principles and patterns, requires the use of inheritance trees (of limited depth), various levels of abstractness, and the exploitation of polymorphism.

Additionally, the correlation analysis revealed some additional metrics that are related to TD principal. For example, one complexity metric (WMC), although significantly correlated to TD has been removed from the model, due to multicollinearity issues. The relation of complexity metrics to TD principal is considered expected, in the sense that the existence of unnecessary complexity in the code of a class implies the need for restructuring the code: the more complex a method, the more TD principal it accumulates. Therefore, our analysis revealed that TD principal is related to metrics that capture the most important low-level quality attributes, namely: coupling, cohesion, inheritance, size, and polymorphism.

Implications to Researchers and Practitioners. The results of this study have provided some interesting implications to practitioners, and future research directions. First, researchers and practitioners can use the proposed model for quantifying the

TD in their projects (see results on RQ_2). This prospect can be interesting for software engineers who: (a) already calculate metrics as part of their quality assurance activities, and (b) want to avoid the installation and parameterization of more complicated tools. In addition to that, based on the finding on discriminative power, researchers and practitioners that are only interested in ranking artifacts, with respect to their TD principal can use an even smaller set of metrics (see results). Regarding future research directions we intend to re-evaluate this model by using an even larger dataset, including a plethora of other well-known open source projects (also smaller ones), that covers more than one programming languages. Additionally, we suggest researchers to further investigate the important metrics (as suggested by our study) with respect to the rest metrics validity criteria described in ISO 1061. Finally, the use of other ways of synthesizing data, e.g., tree-based classifications or Bayes network can be exploited.

VI. Threats to Validity

As already mentioned in this paper, concerning the internal validity of this study, the current assessment of the calculated metrics was based on the source code instead on the design artifacts. These were considered to be equivalent in this study based on the following assumptions, which usually do not stand in practice: (a) the design artifacts are absolutely detailed, and (b) in the foundation of no design drift. Additionally, some metrics that are used as structural proxies are considered in the monetized assessment as well. However, only a small number of these metrics are qualified for our final formula of TD principal prediction. Finally, relating to threats to external validity, we point out two concerns. First, our results depend on the size of the projects in terms of classes. We have chosen large projects for our case study (with more than 300 classes each); therefore our model cannot be generalized to smaller projects. Moreover, our dataset only includes Java projects, which means that the results cannot be generalized to projects developed using other programming languages.

Acknowledgment

This work was financially supported by the action "Strengthening Human Resources Research Potential via Doctorate Research" of the Operational Program "Human Resources Development Program, Education and Lifelong Learning, 2014-2020", implemented from State Scholarship Foundation (IKY) and co-financed by the European Social Fund and the Greek public (National Strategic Reference Framework (NSRF) 2014 – 2020).

References

[1] Z. Li, P. Avgeriou and P. Liang, "A systematic mapping study on technical debt and its management". *Journal of Systems and Software*, *101*, 193-220, 2015.

[2] A. Ampatzoglou, A. Ampatzoglou, P. Avgeriou and A. Chatzigeorgiou, "A Financial Approach for Managing Interest in Technical Debt". In *International Symposium on Business Modeling and Software Design* (pp. 117-133). Springer International Publishing, July, 2015.

[3] J. L. Letouzey and T. Coq, "The SQALE Analysis Model: An Analysis Model Compliant with the Representation Condition for Assessing the Quality of Software Source Code", 2nd International Conference on Advances in System Testing and Validation Lifecycle (VALID' 10), IEEE Computer Society, pp. 43-48, Nice, Paris, 22-27 August 2010.

[4] N. S. R. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull and C. Seaman, "Identification and Management of technical debt: A systematic mapping study," *Inf. Softw. Technol.*, vol. 70, pp. 100–121, Feb. 2016.

[5] B. Curtis, J. Sappidi and A. Szynkarski, "Estimating the principal of an application's technical debt". *IEEE software*, *29*(6), 34-42, 2012.

[6] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, P. Abrahamsson, A. Martini, U. Zdun and K. Systa, "The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study". In 8th International Workshop on Managing Technical Debt (MTD). IEEE Computer Society, 2016.

[7] S. R. Chidamber, D. P. Darcy and C. F. Kemerer, "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis", Transactions on Software Engineering, IEEE Computer Society, 24 (8), pp. 629-639, August 1998.

[8] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment.", *IEEE Transactions on software engineering*, *28*(1), 4-17, 2002.s

[9] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", Journal of Systems and Software, Elsevier, 23 (2), pp. 111-122, November 1993.

[10] M. Riaz, E. Mendes, and E. Tempero, "A systematic review on software maintainability prediction and metrics", 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09), IEEE Computer Society, pp. 367-377, 15-16 October 2009, Florida, USA.

[11] A. van Koten and A.R. Gray, "An Application of Bayesian Network for Predicting Object - Oriented Software Maintainability", Information and Software Technology, Elsevier, 48 (1), pp. 59 – 67, January 2006.

[12] Y. Zhou and H. Leung, "Predicting Object-Oriented Software Maintainability using Multivariate Adaptive Regression Splines", Journal of Systems and Software, Elsevier, 80 (8), pp. 1349 –1361, 2007.

[13] P. Runeson, M. Host, A. Rainer, and B. Regnell, "Case Study Research in Software Engineering: Guidelines and Examples", John Wiley & Sons, 2012.

[14] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou and P. Avgeriou, "Software metrics fluctuation: a property for assisting the metric selection process." *Information and Software Technology*, *72*, 110-124, 2016.

[15] A. Ampatzoglou, A. Gkortzis, S. Charalampidou and P. Avgeriou, "An embedded multiple-case study on oss design quality assessment across domains.", In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on* (pp. 255-258), IEEE, , October , 2013.

[16] I. Griffith and C. Izurieta, "Design pattern decay: the case for class grime.", In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (p. 39), ACM, September, 2014.

[17] D. G. Altman and J. M. Bland, "Statistics notes: the normal distribution." *Bmj*, *310*(6975), 298, 1995.

[18] G. D. Hutcheson and N. Sofroniou, "*The multivariate social scientist: Introductory statistics using generalized linear models". Sage, 1999